Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

A SIMULATION-BASED CONTROLLER FOR DISTRIBUTED DISCRETE-EVENT SYSTEMS WITH APPLICATION TO FLEXIBLE MANUFACTURING

Fernando G. Gonzalez

Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801 USA

Wayne J. Davis

Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801 USA

ABSTRACT

Today, sophisticated discrete-event systems (DES) are being proposed and designed to operate under advanced computer control. In most cases, the complexity of the overall planning and control problem for managing these systems necessitates the use of distributed planning and control architectures. Currently, there is little formalism guiding the construction of these architectures. In this paper, a new coordination architecture, called the Recursive Object-Oriented Control Hierarchy (ROOCH), is employed to address the real-time management of these systems. In addition, a new simulation approach, called Hierarchical Object-Oriented Programmable Logic Simulator (HOOPLS), is introduced for modeling the ROOCH. In order to demonstrate the benefits arising from both the ROOCH architecture and the associated HOOPLS modeling paradigm, the construction of the real-time control architecture for a physical emulator of a flexible manufacturing system (FMS) will be discussed.

1 INTRODUCTION

Over the past decade, the Manufacturing Systems Laboratory (MSL) at the University of Illinois has developed a new conceptual framework for the integrated modeling, scheduling and control of FMSs and other DESs, (see Davis et al. [1993] and Davis et al. [1997]). These developments can now be summarized into three fundamental concepts:

1. The Recursive Object-Oriented Control Hierarchy (ROOCH) architecture which provides a means for defining the component subsystems comprising the overall system,
2. The Hierarchical Object-Oriented Programmable Logic Simulator (HOOPLS) which simulates the developed system architecture by modeling the interactions among the controllers contained within the component subsystems, and
3. The Hierarchical Subsystem Controller (HSC) which provides the generic framework for the intelligent controller contained within each subsystem.

2 THE RECURSIVE OBJECT-ORIENTED CONTROL HIERARCHY (ROOCH)

The ROOCH was developed in collaboration with the Government Electronics Group at Motorola, Inc. and is published in Tirpak et al. [1992]. The ROOCH introduces the coordinated object as the basic building block for modeling an FMS or other DESs, (see Figure 1). Each coordinated object represents a basic hierarchical element where integrated scheduling and control is to be implemented. It is assumed that each coordinated object contains one or more primary unit processes, P_n (n=1,...,N), whose operations are to be scheduled in order to execute tasks upon jobs residing within the coordinated object. Both jobs and supporting resources, (e.g. tools, part kits and processing information), enter the coordinated object through its input port and will eventually exit through the output port. The jobs and supporting resources are assumed to be under the control of the coordinated object while they reside within the coordinated object.

As stated above, the coordinated object is the basic hierarchical element where scheduling and control occurs. To this end, each coordinated object contains a Hierarchical System Coordinator (HSC) to perform the concurrent functions of scheduling the allocation of the subordinate processes and controlling the flow of the entities in order to implement the developed schedule. In order to manage the flow of entities, it is further assumed that each coordinated object contains one or more material handling systems (MHSs) that are capable of moving entities among the various queues contained within the coordinated object. (The MHSs can be viewed as supporting unit processes since they do not execute processing steps that physically transform the product.) It is absolutely essential that these MHSs exist within the coordinated objects as each coordinated object must be able to effect the entity movement required to implement its schedule. The recursive nature of the ROOCH arises from the fact that any subordinate subsystem within a coordinated object can also be another coordinated object. This recursive feature permits the modeler to construct the ROOCH with as many hierarchical levels as are needed to model the FMS or DES.

The HSC contained within a coordinated object represents only types of control objects that will be defined for the ROOCH. In general, the controllers contained within the ROOCH can be classified as one of three types of control objects. A coordination node represents the HSC associated with coordinated object. The primary distinction here is that the coordinated object must contain unit processes that are capable of performing processing tasks upon the job entities. A transportation node also represents the HSC within a coordinated object. However, in this case, the coordinated object contains transport processes only. An example of a transport node is the primary controller to a MHS. Finally, a process node can only execute a processing or transport instruction. In general, a process node does not perform planning.

In order to prevent confusion, let us reiterate the difference between a coordinated object and coordination node. The coordinated object is a modeling element for a defined subsystem. It must contain a control object which can either be a coordination node or a transportation node depending on whether or not the coordinated object contains subsystems (or processes) which can perform processing steps.

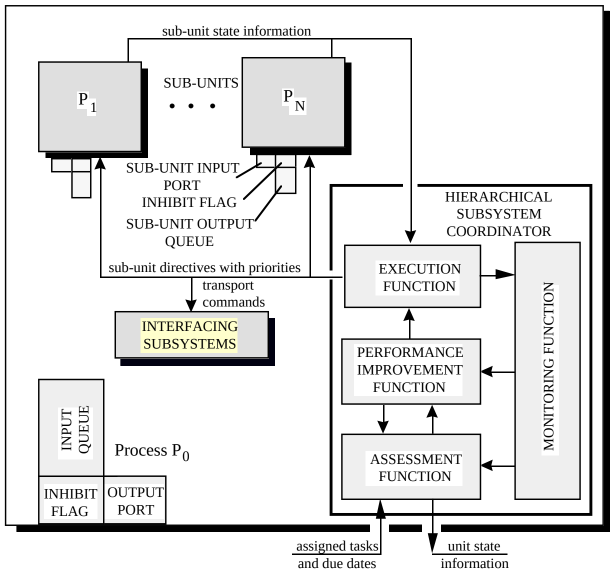

Figure 1. Schematic of the Coordinated Object: The Basic Module for Planning and Control

3 THE HIERARCHICAL OBJECT-ORIENTED PROGRAMMABLE LOGIC SIMULATOR (HOOPLS)

HOOPLS is a simulation methodology based upon the belief that in order to accurately model a system with a distributed control architecture, the interactions among the controllers must be considered within the model. The single most important characteristic of the HOOPLS methodology, and what separates it from other object-oriented research efforts in simulation, is its focus upon modeling the flow of messages among the included controllers. The modeling of the controllers must also recognize the manner in which each element of the coordinated object can modify the state of a given entity. The coordinated object, itself, can reassign the ownership of an entity only by assigning it to a subordinate subsystem for the implementation of a specified set of tasks. The MHSs (which may also be coordinated objects) included within the coordinated object can only change the location of the entity. Finally, it is the primary unit processes which execute the assigned processing tasks that physically modify the state of an entity. HOOPLS explicitly considers these constraints when it defines the state-transition mechanisms for the evolution of the DES.

4 THE FMS EMULATOR

In order to develop an educational and research laboratory for the coordination of DESs, the Manufacturing Systems Laboratory at the University of Illinois has constructed an FMS emulator (see Davis et al. [1994]). To insure safety and economy, all physical processing has been omitted. We have also constructed physical analogs for the material handling systems (MHSs). In developing this emulator, we have demonstrated the ability for HOOPLS to serve as a computer-aided tool for designing the emulator’s control architecture.

The schematic for the constructed FMS emulator is depicted in Figure 2. Figure 3 provides a photograph of the emulator. The emulator has four Processing Centers, numbered 1 through 4. Each Processing Center (PC), shown in Figure 4, is a coordinated object containing one primary subordinate processing resource (the Unit Process) and a dedicated MHS.

The PC’s MHS is represented by a carousel with six electromagnets which can hold the jobs (represented by color-coded steel washers) that reside in either the input or output queue of the PC. The movement of the carousel is controlled by a dedicated Programmable Logic Controller (PLC). The PC's controller is a UNIX workstation which is placed upon a rostrum atop each PC. This controller is connected to the MHS PLC via a dedicated RS 232 link, and its display provides the summary status for the MHS and the subordinate unit process. Because each PC does contain a unit process and is a coordinated object, the PC's controller represents a coordination node.

Another subordinate subsystem is the Fixturing Center (FC) which also represents a coordinated object. The structure and emulation of the FC is very similar to that of a PC. The FC has a dedicated MHS consisting of a primary carousel capable of holding sixteen jobs and two smaller carousels for loading and unloading jobs from the AGVs. The movement of these carousels is controlled by a dedicated PLC. The FC

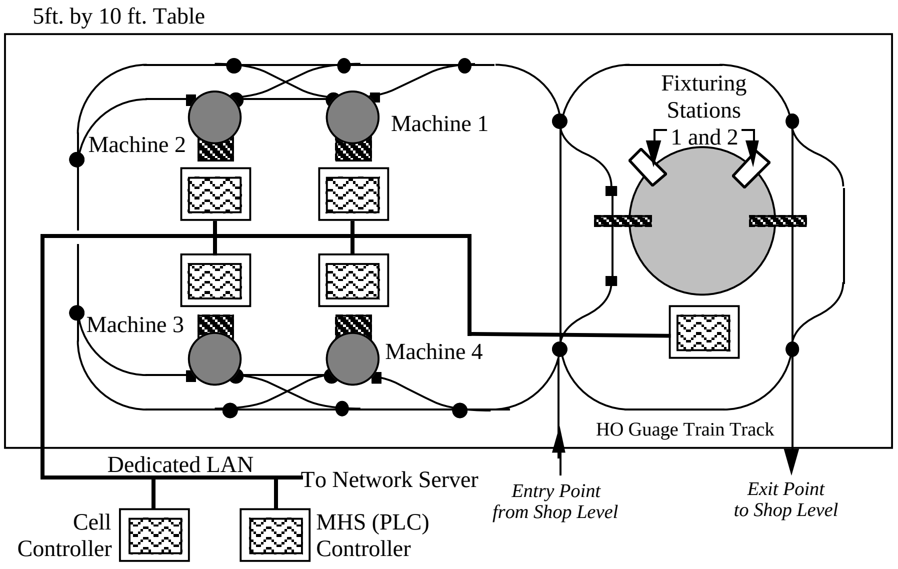

Figure 2. Schematic for the Layout of the Physical Emulator

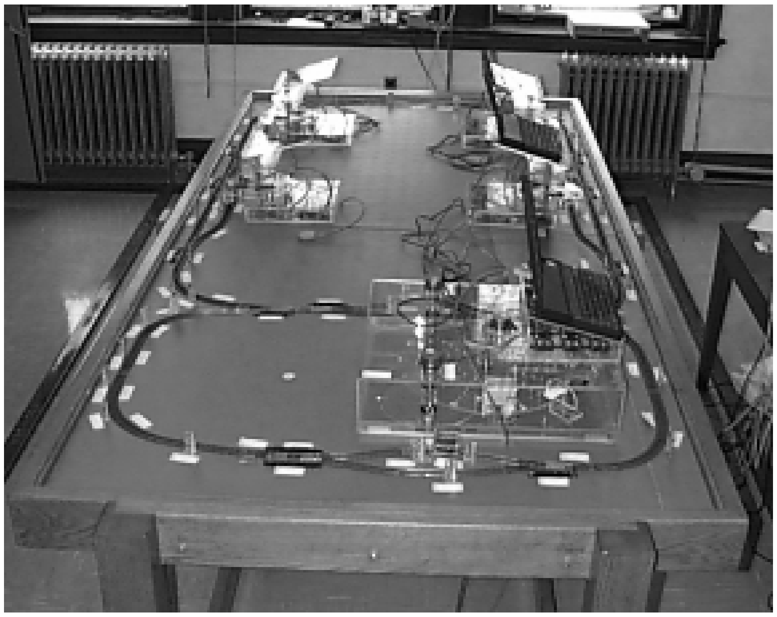

Figure 3. A Photograph of the Physical Emulator

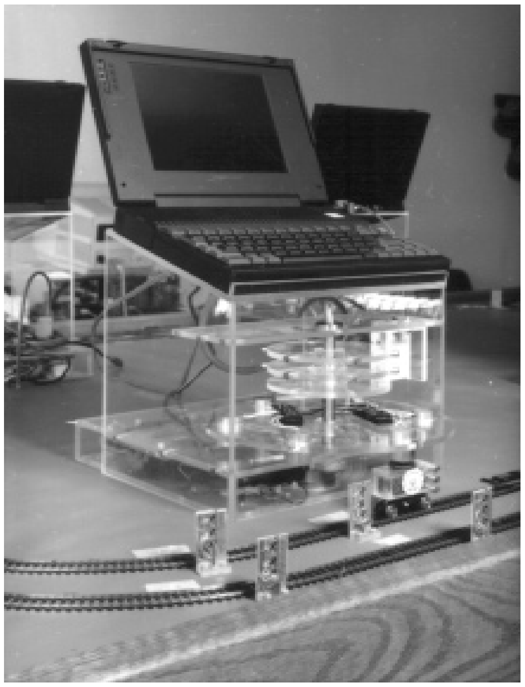

Figure 4. (right) Picure of the Processing Center. Note that the HO-scale train is parked in front of processing center.

has two fixturing positions which represent the subordinate unit processes. A dedicated laptop UNIX workstation provides the real-time status information for each fixturer. This same workstation also issues control messages to the dedicated PLC for the FC's MHS. Because the FC contains two unit processes and is a coordinated object, the FC's controller represents another coordination node.

The final subordinate process is the Cell MHS. An Automated Guided Vehicle (AGV) system is employed as the cell's MHS, and the AGVs are emulated with HO-scale electric trains. (A train is shown in front of the PC pictured in Figure 4.) The train layout is diagrammed in Figure 2 and pictured in Figure 3. In this layout, there are over forty track segments which can be individually powered. Sensory switches, also shown in Figure 4, are provided on each track segment in order to detect the presence of an AGV. A Petri net has been developed to insure that no more than one AGV ever occupies a single track segment at a time.

A dedicated PLC receives directives from the MHS controller to move a given AGV from one location to another using the incorporated Petri net logic. The dedicated PLC returns the location of each AGV as it enters each track segment to the MHS controller. The MHS controller is also responsible for determining the order in which the pending material handling transfers will be processed and which AGV will be assigned to complete each requested transfer. Although the MHS is a coordinated object, it does not contain unit processes that are capable of executing processing instructions upon a job entity. Therefore, the MHS controller represents a transport node.

The cell controller is implemented by another UNIX workstation. It is connected to each of the subordinate PCs, FC and MHS controllers (and a network server) via an ether network. Various commands and feedback information flow across this network. The role of the cell Controller is to orchestrate the flow of all entities types (jobs as well as supporting resources) among the subordinate processes within the cell. The cell, in this case, represents another coordinated object whose subsystems are also coordinated objects. Five of these subsystems, the four PCs and the FC, are capable of executing processing tasks. Therefore, the cell controller represents a coordination node.

5 THE CONTROL ARCHITECTURE

The model for the physical emulator has now been decomposed into a set of coordinated objects using the ROOCH decomposition architecture. The complete model consists of 25 control objects which includes control objects for the coordinated objects defined above and the controllers for the basic unit and transport processes that are contained with each coordinated object (see Table 1). Each control object is implemented as an independent controller which communicates with its supervisor and its subordinates using communication messages defined within HOOPLS. The communication employs a local area network (LAN) connecting seven computers. The cell controller is situated on one computer while each of the cell's six subordinate coordinated objects' controllers is situated on its own computer.

Each controller was implemented using a simulation software tool that we have developed (see Gonzalez [1996]). The software tool provides a collection of C++ objects whose integration with the controller object's model permits each controller to be executed as an independent distributed object. Thus, each of the 25 controller objects has its own instantiation of the simulation tool. This implies that there are 25 independent copies of the simulation tool, each executing its own model concurrently with the others. The only thing that coordinates the controllers' actions into a unified control architecture is the communication that occurs during their interactions.

Figure 5 shows the ROOCH decomposition of the model. Note that when the developed model is employed to control the FMS, the supervisor represents an independent agent that assigns tasks to the cell controller. It is not part of the modeled control architecture per se. The PLC controllers that are shown at the bottom level are not implemented in the control model either because they are implemented by the actual PLCs within the emulator. Although logically they work the same way as the other controller models, the PLC does not support our software tool. Instead, the controllers must be implemented in Dynamic C, the C compiler that resides in the PLC. Since the actual code used to build the models in the PLC cannot be run on the workstations, their actions had to be modeled for simulation purposes.

In summary, the submodels that are presented between the two horizontal lines in Figure 5 are the employed controllers for the FMS emulator. When the model is being employed to control the FMS, the process controllers at the bottom of Figure 4 will be implemented by the control code contained within the dedicated PLCs that are managing the processes. When the model is being employed in the simulation mode, additional models for the control objects representing the PLCs will be added to the model. Finally, during the control mode, a supervisor for the cell will be provided where tasks can be inputted into the cell. In the simulation mode, we employ a typical creation process which models the arrival of job entities into the cell.

Hence, the HOOPLS simulation model for the basic ROOCH has been designed such that the same model can be used both to simulate and to control the system. The

Table 1. The Control Objects

Control Object's Name	Type of Node	Nodes Title	Nodes Task
fmssup	Coordinating Node	Cell supervisor	Assigns jobs to the cell controller.
fmscell	Coordinating Node	Cell controller	Coordinates the cells 5 processing centers and the AGV.
fscell	Coordinating Node	Fixturing station's cell controller	Coordinates the Fixturing stations 2 processes and its MHS.
fsproc1	Processing Node	Fixturing station 1 process controller	Represents the Fixturing station 1 processor.
fsproc2	Processing Node	Fixturing station 2 process controller	Represents the Fixturing station 2 processor.
fsplc	Not a Coordinated Object	Fixturing station's PLC driver	Models the Fixturing station's PLC driver.
cellmhs	Transport Node	Cell's MHS controller	Controls the MHS of the cell.
mhsplc	Not a Coordinated Object	Cell's MHS PLC driver	Represents the cell's MHS driver
pc1cell	Coordinating Node	Processing center 1 cell controller	Coordinates the processing center 1 processor and its MHS.
pc1proc	Processing Node	Processing center 1 process controller	Represents the processing center 1 processor
pc1mhs	Transport Node	Processing center 1 MHS controller	Controls the MHS for processing center 1
pc1plc	Not a Coordinated Object	Processing center 1 PLC driver	Models the processing center 1 PLC driver.

Note that the control objects for PC2, PC3 and PC4 are identical to those of PC1 and are not repeated in the table.

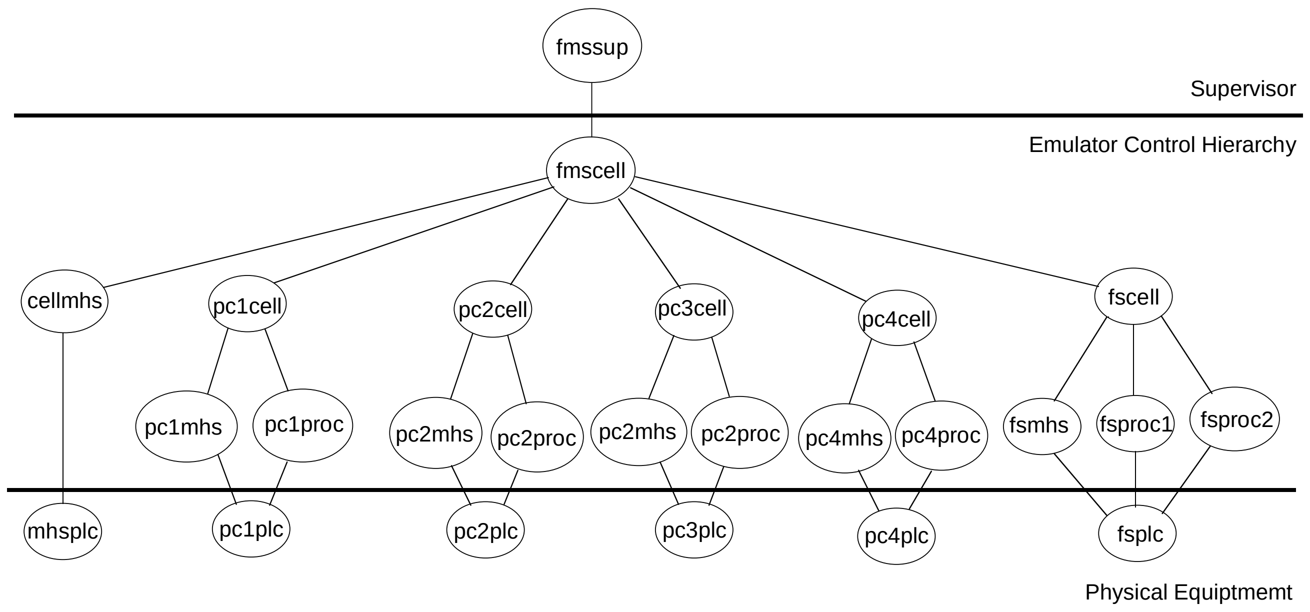

Figure 5. The ROOCH Decomposition of the Model Used to Control the Emulator

model is entirely characterized through an exhaustive definition of the control messages that are issued or received by each controller contained within the ROOCH (see Table 2) and the state transition mechanisms which will occur upon the receipt of a control message. These state transition mechanisms, in turn, determine the subsequent control messages that will be issued by the receiving controller. Due to limited space, the state transition mechanisms are not presented.

5.1 The Communication Messages

Recall that the single most important characteristic of the HOOPLS methodology is its focus upon modeling the flow of control messages. The dedicated control object within the submodel for a given coordinated object simply generates the appropriate response based upon the message it receives. The dynamics that occur within the coordinated object must also be explained. The submodels for the coordinated objects must also model the flow of entities within each coordinated object. These submodels are responsible for explaining the state transition mechanisms of the coordinated objects. When certain events occur, they cause the controller object within the modeled coordinated object to issue its response to an incoming control message. In short, an incoming control message initiates a sequence of state transitions to occur resulting in an event which causes the controller object to respond. This response can be viewed as an output from the coordinated object. The HOOPLS methodology is not concerned with the methodology employed to construct submodels for the coordinated objects so long as the overall coordination among the submodels is implemented by modeling the controller interactions.

For the modeling of the FMS emulator, the employed control messages are given in Table 2. All of these messages, except for the ExecuteTask and TaskExecuted, pertain to moving entities within the manufacturing cell. In most cases, the entity is being moved from one MHS to another. In order to transfer the entity between two MHSs, one of the two MHS must perform the loading while the other simply waits at an input or output port until the loading is complete. In all cases, the MHS that simply waits must be at the port before the loading MHS starts loading. This means the response from the simple MHS must be received before the command for loading MHS can be issued. In order to allow both MHSs to move to the input or output port at the same time, the AnticipateItem message is issued to the loading MHS so that it can move to the port, but not load. The simple MHS is issued the regular command to move. Since the simple MHS does not know when the loading is completed, the ResourceFree command must be employed in order to tell the simple MHS that the loading is done, and it is free to move again.

Table 2. The HOOPLS Messages

From	To	Message	Optional Operand	Required Operand
CN	CN	AcceptItem	*where*	*serial number*
CN	CN	ItemAccepted		*serial number*
CN	CN	ReturnItem	*where*	*serial number*
CN	CN	ItemReturned		*serial number*
CN	TN	ResourceFree		*serial number*
CN	CN	ExecuteTask	*queue*	*serial number*
CN	PN	ExecuteTask	*Task*	*serial number*
PN,CN	CN	TaskExecuted		*serial number*
CN	TN	PickUpItem	*where*	*serial number*
TN	CN	ItemPickedUp		*serial number*
CN	TN	DeliverItem	*where*	*serial number*
TN	CN	ItemDelivered		*serial number*
CN	CN,TN	AnticipateItem	*where*	*serial number*
TN,PN	PLC	*message*		NULL

where	- the name of the location.
task	- The task. Can be a string or a list of strings.
serial number	- The serial number of the job (job number).
message	- Any arbitrary message.
CN	- Coordinating Node.
PN	- Processing Node.
TN	- Transport Node.
PLC	- Programmable Logic Controller

The supervisor is assumed to know the limitations of its subordinate controllers. It will not request that a subordinate controller execute an operation that will put the subordinate in an erroneous state. That is, the supervisor knows how many parts can fit into a particular machine and will not request that station's controller to accept a new job if the subordinate has no available capacity at the time. In this manner, the subordinate controller need not check to see if it has the capacity to handle a request from its supervisor. However, in cases where the machine can be put into an erroneous state causing physical damage or interrupting normal response, the subordinate controller can screen the request. These redundant checks provide an extra measure of safety.

6 THE REAL-TIME SIMULATION

The HOOPLS executive function consists of a global event list and a message relay (see Figure 6). When a submodel is executed, the message relay receives all of the messages that are outputted by the submodel currently being executed. The messages are inserted into the rear of the message queue that is used to implement the message relay. Once the submodel finishes execution, the HOOPLS function removes the message that is at the front of the message queue and gives it to the submodel to which the message is addressed. Upon receiving the message, the submodel then executes the appropriate event to process the message. Since further messages may be generated by the execution of this event, these messages are again inserted into the rear of the message queue. The above process is repeated until there are no messages in the message queue.

After the message relay is emptied, the main HOOPLS function pulls the next event from the global event list. The events in this list have two pieces of information, the place and the time that the next event will occur. The information about the type of event and other details about the event are stored within a local event list contained within the submodel. Hence, whenever a submodel stores an event on its dedicated event calendar, an abbreviated copy of the event is also stored on the global event calendar. Since all submodels schedule events in this manner, the global event list always maintains a chronological order of the events that are to occur in all of the submodels. The main HOOPLS function updates its clock with the time in the global event and calls the appropriate submodel with the new current time. The submodel, knowing that it has the next event, pulls the next event off its local event list and executes it. After the event is processed, including the scheduling of future events (on both the local and global event calendars) and response messages are created and sent to the message relay, program control is then returned to the HOOPLS function. The main HOOPLS function starts its cycle over again by checking the message relay for new messages.

The employment of global and local event lists provides an essential means for synchronizing the events that are being processed in the submodels. Currently other means are being explored in order to synchronize the execution of events. Other have also been employing simulation to control manufacturing systems. Peters et al. [1996] have adapted ARENA® to control their experimental FMS. However, their development is limited to one hierarchical level only where the supervisor, the cell controller, manages a set of subordinate processes. They too must include special events which send messages to the controllers. Narayanan et al. [1992] and Govindaraj [1993] have also developed simulation tools which share many of the properties of HOOPLS. They too model the controller interactions. From this set of independent development effort, the need to model controller interactions is becoming evident.

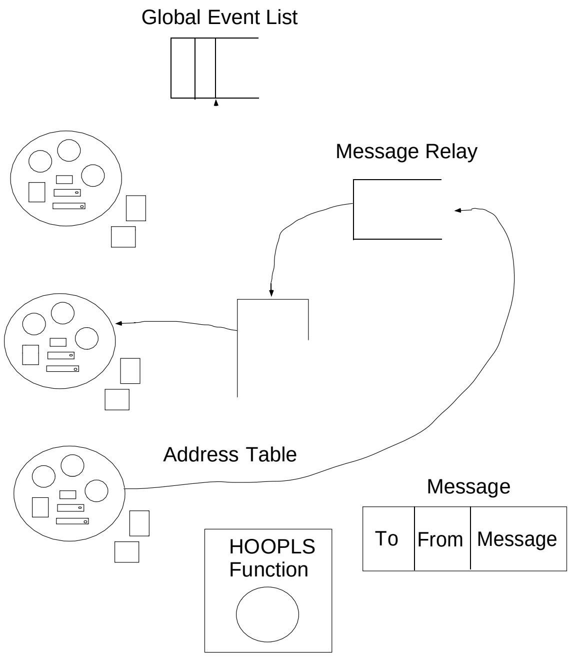

Figure 6. An Example of a HOOPLS Simulation of a Controller Decomposed into Three Subordinate Controllers. Each of the three subordinate controllers are included in the simulation with no modification. The bottom controller sends a message to the message relay addressed to the middle controller. The message relay sends the message to the middle controller. While the middle controller was executing this message, it executed a DELAY event which scheduled an event into the global event list. At sometime later, the global event is pulled off the global event list and the middle controller is "awakened" to execute this event.

7 FUTURE WORK

The next step in this research is to develop the Hierarchical Subsystem Coordinator (HSC) which will perform the real-time scheduling and control functions for each coordinated object. Our immediate research plans will address the development of a real-time scheduling capability for the cell controller. The implemented HOOPLS simulation presented in this paper will then be employed within this scheduler to perform the real-time simulations which can project future system performance to assist in scheduling process (see Davis et al. [1996] and Davis [1998]). The cell controller will still control the system in the manner discussed in this paper, but it will able to dynamically change its control strategy to implement the current plan developed by the real-time scheduler. The physical FMS emulator will continue to serve as the testbed for the algorithmic developments.

REFERENCES

Davis, W. J., D. Setterdahl, J. Macro, V. Izokaitis and B. Bauman. 1993. Recent Advances in the Modeling, Scheduling and Control of Flexible Automation. *Proc. of the 1993 Winter Simulation Conference*, eds. G.W. Evans, M. Mollaghasemi, E.C. Russell and W.E. Biles, 143-155, The Society for Computer Simulation, San Diego, CA.

Davis, W.J., B. Bauman, J. Macro and D. Setterdahl. 1994. Constructing an Emulator for Research and Education in the Control of Flexible Automation. *Proceedings of the ORSA Technical Section on Manufacturing Management Conf.*, eds. J. Buzacott and C.A. Yano, 151-157.

Davis, W. J., J. G. Macro, A. L. Brook, M. S. Lee, G. S. Zhou. 1996. Developing a Real-Time Emulation / Simulation Capability for the Control Architecture to the RAMP FMS. *Proc. of the 1996 Winter Simulation Conference*, eds. J. M. Charnes, D. M. Morrice, D. T. Brunner, J. J. Swain, 171-178.

Davis, W.J., J. Macro and D. Setterdahl. 1997. An Integrated Methodology for the Modeling, Scheduling and Control of Flexible Automation. To appear in the *J. on Robotics and Intelligent Control.*

Davis, W.J. 1998. Real-Time Simulation: The Need and the Evolving Research Requirement. To appear in the *Simulation Handbook*, ed. J. Banks, 67 pages, Wiley, New York.

Govindaraj, T., L.F. McGinnis, C.M. Mitchell, D. A. Bodner, S. Narayanan and U. Sreekanth. 1993. OOSIM: A Tool for Simulating Modern Manufacturing Systems. *Proc. of the 1993 National Science Foundation Grantees in Design and Manufacturing Conference*, 1055-1062.

Gonzalez, F. G. 1996. A Simulation-Based Controller Builder for Flexible Manufacturing Systems. *Proc. of the 1996 Winter Simulation Conference*, eds. J. M. Charnes, D. M. Morrice, D. T. Brunner, J. J. Swain, 1068-1075.

Narayanan, S.D., A. Bodner, U. Sreekanth, S.J. Dilley, T. Govindaraj, L.F. McGinnis and C.M. Mitchell. 1992. Object-Oriented Simulation to Support Operator Decision Making in Semiconductor Manufacturing. *Proc. of the 1992 Intl. Conf on Systems, Man and Cybernetics*, 1510-1519, The Institute of Electrical and Electronics Engineers, Piscataway, NJ.

Peters, B.A., J.S. Smith, J. Curry and C. LaJimodiere. 1996. Advanced Tutorial - Simulation Based Scheduling and Control, *Proc. of the 1996 Winter Simulation Conference*, eds. J.M. Charnes, D.J. Morrice, D.T. Brunner and J.J. Swain, 194-198, The Society for Computer Simulation, San Diego, CA.

Tirpak, T.M., S.M. Daniel, J.D. LaLonde, and W.J. Davis. 1992. A Fractal Architecture for Modeling and Controlling Flexible Manufacturing Systems, *IEEE Trans. on Systems, Man and Cybernetics*, 22(5), 564-567.

AUTHOR BIOGRAPHIES

FERNANDO G. GONZALEZ received his Ph.D. in Electrical and Computer Engineering at the University of Illinois. Earlier, he received his B.S. in computer science and M.S. in electrical engineering at Florida International University. His current research addresses the real-time management of discrete-event systems. He was a recipient of an NSF Support for Under Represented Groups in Engineering (SURGE) Fellowship and a GTE Minority Fellowship. Dr. Gonzalez is currently employed as a post-doctorate researcher to further improve the emulator discussed in this paper.

WAYNE J. DAVIS is a Professor of General Engineering at the University of Illinois at Urbana-Champaign. He received his degrees in Engineering Sciences from Purdue University. His current research addresses the intelligent control architectures for large-scale discrete-event systems. In this effort, he is collaborating with the Intelligent Systems Division at the National Institute of Standards and Technology. He is also developing several new simulation languages in order to support this development.